Contents

1 Data Structures

1.1 BIT

```
//1-based

void update(int n, int idx, int v){
  while(id <= n) tree[idx] += v, idx += idx & (-idx);
}

int query(int id){
  int sum = 0;
  while(idx > 0) sum += tree[idx], idx -= idx & (-idx);
  return sum;
}
```

1.2 CHT

```
/*
If m is decreasing:
  for min : bad(s-3, s-2, s-1), for max : bad(s-1, s-2, s-3)
If m is increasing:
  for max : bad(s-3, s-2, s-1), for min : bad(s-1, s-2, s-3)
If x isn't monotonic, then do Ternary Search or keep intersections and do binary search
*/

struct CHT{
  vector<ll> m, b;
  int ptr = 0;
  bool bad(int l1, int l2, int l3) { // returns intersect(l1, l3) <= intersect(l1, l2)
    return 1.0 * (b[l3] - b[l1]) * (m[l1] - m[l2]) <= 1.0 * (b[l2] - b[l1]) * (m[l1] - m[l3]);
  }

  void insert_line(ll _m, ll _b) {
    m.push_back(_m);
    b.push_back(_b);
    int s = m.size();
    while(s >= 3 && bad(s-3, s-2, s-1)) {
      s--;
      m.erase(m.end()-2);
      b.erase(b.end()-2);
    }
  }
  ll f(int i, ll x) { return m[i]*x + b[i]; }

  ll eval(ll x) {
    if(ptr >= m.size()) ptr = m.size()-1;
    while(ptr < m.size()-1 && f(ptr+1, x) > f(ptr, x)) ptr++;
    return f(ptr, x);
  }
};
```

1.3 Dynamic CHT

```
const ll is_query = -LLONG_MAX;
struct Line {
  ll m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != is_query) return m < rhs.m;
    const Line* s = succ();
    if (!s) return 0;
    ll x = rhs.m;
    return b - s->b < (s->m - m) * x;
  }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
  bool bad(iterator y) {
    auto z = next(y);
    if (y == begin()) {
      if (z == end()) return 0;
      return y->m == z->m && y->b <= z->b;
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b <= x->b;
    //may need to use __int128 instead of ld if supported
    return ld(x->b - y->b)*(z->m - y->m) >= ld(y->b - z->b)*(y->m - x->m);
  }
  void insert_line(ll m, ll b) {
    auto y = insert({ -m, -b }); //change here for min
    if (bad(y)) { erase(y); return; }
    while (next(y) != end() && bad(next(y))) erase(next(y));
    y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
    while (y != begin() && bad(prev(y))) erase(prev(y));
    if(y != begin()) prev(y)->succ = [=] { return &*y; };
  }
  ll eval(ll x) {
    auto l = *lower_bound((Line) { x, is_query });
    return -(l.m * x + l.b); //change here for min
  }
} hull;
```

1.4 FFT

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef long double ld;

struct cplx {
  ld a, b;

  cplx (ld a = 0, ld b = 0) : a(a), b(b) {}

  const cplx operator + (const cplx &c) const {
    return cplx(a + c.a, b + c.b);
  }

  const cplx operator - (const cplx &c) const {
    return cplx(a - c.a, b - c.b);
  }

  const cplx operator * (const cplx &c) const {
    return cplx(a * c.a - b * c.b, a * c.b + b * c.a);
  }

  const cplx conj() const {
    return cplx(a, -b);
  }
};

const ld PI = acosl(-1);
const int MOD = 1e9 + 7;
const int N = (1 << 20) + 5;

int rev[N]; cplx w[N];

void prepare (int &n) {
  int sz = __builtin_ctz(n);
  for (int i = 1; i < n; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (sz - 1));
  w[0] = 0, w[1] = 1, sz = 1;
  while (1 << sz < n) {
    ld ang = 2 * PI / (1 << (sz + 1));
    cplx w_n = cplx(cosl(ang), sinl(ang));
    for (int i = 1 << (sz - 1); i < (1 << sz); ++i) {
      w[i << 1] = w[i], w[i << 1 | 1] = w[i] * w_n;
    } ++sz;
  }
}

void fft (cplx *a, int n) {
  for (int i = 1; i < n - 1; ++i) {
    if (i < rev[i]) swap(a[i], a[rev[i]]);
  }
  for (int h = 1; h < n; h <<= 1) {
    for (int s = 0; s < n; s += h << 1) {
      for (int i = 0; i < h; ++i) {
        cplx &u = a[s + i], &v = a[s + i + h], t = v * w[h + i];
        v = u - t, u = u + t;
      }
    }
  }
}

static cplx f[N], g[N], u[N], v[N];

void multiply (int *a, int *b, int n, int m) {
  int sz = n + m - 1;
  while (sz & (sz - 1)) sz = (sz | (sz - 1)) + 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) f[i] = cplx(i < n ? a[i] : 0, i < m ? b[i] : 0);
  fft(f, sz);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (f[i] * f[i] - (f[j] * f[j]).conj()) * cplx(0, -0.25);
    f[j] = x, f[i] = x.conj();
  }
  fft(f, sz);
  for(int i = 0; i < sz; ++i) a[i] = f[i].a / sz + 0.5;
}
```

```
inline void multiplyMod (int *a, int *b, int n, int
↪  m) {
  int sz = 1;
  while (sz < n + m - 1) sz <<= 1;
  prepare(sz);
  for (int i = 0; i < sz; ++i) {
    f[i] = i < n ? cplx(a[i] & 32767, a[i] >> 15) :
↪   cplx(0, 0);
    g[i] = i < m ? cplx(b[i] & 32767, b[i] >> 15) :
↪   cplx(0, 0);
  }
  fft(f, sz), fft(g, sz);
  for (int i = 0; i < sz; ++i) {
    int j = (sz - i) & (sz - 1);
    static cplx da, db, dc, dd;
    da = (f[i] + f[j].conj()) * cplx(0.5, 0);
    db = (f[i] - f[j].conj()) * cplx(0, -0.5);
    dc = (g[i] + g[j].conj()) * cplx(0.5, 0);
    dd = (g[i] - g[j].conj()) * cplx(0, -0.5);
    u[j] = da * dc + da * dd * cplx(0, 1);
    v[j] = db * dc + db * dd * cplx(0, 1);
  }
  fft(u, sz), fft(v, sz);
  for(int i = 0; i < sz; ++i) {
    int da = (ll) (u[i].a / sz + 0.5) % MOD;
    int db = (ll) (u[i].b / sz + 0.5) % MOD;
    int dc = (ll) (v[i].a / sz + 0.5) % MOD;
    int dd = (ll) (v[i].b / sz + 0.5) % MOD;
    a[i] = (da + ((ll) (db + dc) << 15) + ((ll) dd
↪   << 30)) % MOD;
  }
}

int main(){
  int a[6] = {1, 1, 2};
  int b[6] = {1, 1, 1};
  multiply(a, b, 3, 3);

  for(int i = 0; i < 6; i++) cout << i << ' ' <<
↪   a[i] << '\n';
}
// (2x^2 + x + 1)(x^2 + x + 1)

// (2x^4 + 2x^3 + 2x^2 + x^3 + x^2 + x + x^2 + x +
↪   1)
// (2x^4 + 3x^3 + 4x^2 + 2x + 1)
```

1.5 FWHT

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int N = 1 << 20;

// apply modulo if necessary

void fwht_xor (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x + y, a[j + h] = x - y;
        if (dir) a[j] >>= 1, a[j + h] >>= 1;
      }
    }
  }
}

void fwht_or (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = x, a[j + h] = dir ? y - x : x + y;
      }
    }
  }
}

void fwht_and (int *a, int n, int dir = 0) {
  for (int h = 1; h < n; h <<= 1) {
    for (int i = 0; i < n; i += h << 1) {
      for (int j = i; j < i + h; ++j) {
        int x = a[j], y = a[j + h];
        a[j] = dir ? x - y : x + y, a[j + h] = y;
      }
    }
  }
}

int n, a[N], b[N], c[N];

int main() {
  n = 1 << 16;
  for (int i = 0; i < n; ++i) {
    a[i] = rand() & 7;
    b[i] = rand() & 7;
  }
  fwht_xor(a, n), fwht_xor(b, n);
  for (int i = 0; i < n; ++i) {
    c[i] = a[i] * b[i];
  }
  fwht_xor(c, n, 1);
  for (int i = 0; i < n; ++i) {
    cout << c[i] << " ";
  }
  cout << '\n';
  return 0;
}
```

1.6 Li Chao Tree

```
//Li Chao Tree for minimum case

struct Line {
  ll m, c;
  Line(ll m = 0, ll c = 0) : m(m), c(c) {};
  inline ll f(ll x) { return m * x + c; }
};

Line tree[4 * N];

void insert(int rt, int l, int r, Line v){
  if(l == r){
    if(tree[rt].f(l) > v.f(l)) tree[rt] = v;
↪   //change to < for max
    return;
  }
  int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
  bool lft = v.f(l) < tree[rt].f(l); //change to >
↪   for max
  bool mid = v.f(m) < tree[rt].f(m); //change to >
↪   for max

  if(mid) swap(tree[rt], v);
  if(lft != mid) insert(lc, l, m, v);
  else insert(rc, m + 1, r, v);
}

ll query(int rt, int l, int r, int x){
  if(l == r) return tree[rt].f(x);
  int m = l + r >> 1, lc = rt << 1, rc = lc | 1;
  //replace min with max for max query
  if(x <= m) return min(tree[rt].f(x), query(lc, l,
↪   m, x));
  else return min(tree[rt].f(x), query(rc, m + 1,
↪   r, x));
}
```

1.7 Linear Sieve

```
const int PRIME_SZ = ;
int prime[PRIME_SZ], prime_sz;
bitset<N> mark;

void sieve(){
  for(int i = 2; i < N; ++i){
    if(!mark[i])prime[prime_sz++] = i;
    for(int j = 0; j < prime_sz and i * prime[j] <
↪   N and (!j or j and i % prime[j - 1]); ++j){
      mark[i * prime[j]] = true;
    }
  }
}
```

1.8 Matrix Expo

```
struct matrix {
  ll mat[100][100]; // make this as small as
↪   possible
  int dim;
  matrix(){};
  matrix(int d){
    dim = d;
    for(int i = 0; i < dim; i++)
      for(int j = 0; j < dim; j++) mat[i][j] = 0;
  }
  matrix operator *(const matrix &mul){
    matrix ret = matrix(dim);
    for(int i = 0; i < dim; i++){
      for(int j = 0; j < dim; j++){
        for(int k = 0; k < dim; k++){
          ret.mat[i][j] += mat[i][k] *
↪   mul.mat[k][j];
          ret.mat[i][j] %= MOD;
        }
      }
    }
    return ret;
  }
  matrix operator + (const matrix &add){
    matrix ret = matrix(dim);
    for(int i = 0; i < dim; i++){
      for(int j = 0; j < dim; j++){
        ret.mat[i][j] = mat[i][j] + add.mat[i][j];
        ret.mat[i][j] %= MOD;
      }
    }
    return ret ;
  }
  matrix operator ^(int p){
    matrix ret = matrix(dim);
    matrix m = *this;
    for(int i = 0;i < dim; i++) ret.mat[i][i] = 1;
    while(p){
      if(p & 1) ret = ret * m;
      m = m * m; p >>= 1;
    }
    return ret;
  }
};
```

1.9 Trie

```
const int MAX = 1e5 + 5, ALPHA = 10; // total
↪   characters, alphabet size
const char START = '0'; // first letter in alphabet

inline scale(char c){ return c - START; }

struct Trie {
  int root, nodes, nxt[MAX][ALPHA], finished[MAX];

  Trie(){
    root = nodes = 1;
    memset(nxt, 0, sizeof nxt);
  }

  void insert(string s){
    int cur = root;
    for(auto c : s){
      if(!nxt[cur][scale(c)]) nxt[cur][scale(c)] =
↪   ++nodes;
      cur = nxt[cur][scale(c)];
    }
    finished[cur]++;
  }

  bool find(string s){
    int cur = root;
    for(auto c : s){
      if(!nxt[cur][scale(c)]) return false;
      cur = nxt[cur][scale(c)];
    }
    return finished[cur];
  }

  void erase(string s){ // may need to call find()
↪   before
    int cur = root;
    for(auto c : s) cur = nxt[cur][scale(c)];
    finished[cur]--;
  }
};
```

2 Geometry

2.1 Convex Hull

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair <ll, ll> point;

#define x first
#define y second

inline ll area (point a, point b, point c) {
  return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) *
↪   (c.x - a.x);
}

vector <point> convexHull (vector <point> p) {
  int n = p.size(), m = 0;
  if (n < 3) return p;
  vector <point> hull(n + n);
  sort(p.begin(), p.end());
  for (int i = 0; i < n; ++i) {
    while (m > 1 and area(hull[m - 2], hull[m - 1],
↪   p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  for (int i = n - 2, j = m + 1; i >= 0; --i) {
    while (m >= j and area(hull[m - 2], hull[m -
↪   1], p[i]) <= 0) --m;
    hull[m++] = p[i];
  }
  hull.resize(m - 1); return hull;
}

int main() {
  int n; cin >> n;
  vector <point> p(n);
  for (auto &it : p) scanf("%lld %lld", &it.x,
↪   &it.y);
  vector <point> hull = convexHull(p);
  for (auto it : hull) printf("%lld %lld\n", it.x,
↪   it.y);
  return 0;
}
```

2.2 Minimum Enclosing Circle

```
// Expected runtime: O(n)
// Solves Gym 102299J

#include <bits/stdc++.h>

using namespace std;

typedef long double ld;
typedef pair <ld, ld> point;

#define x first
#define y second

point operator + (const point &a, const point &b) {
  return point(a.x + b.x, a.y + b.y);
}

point operator - (const point &a, const point &b) {
  return point(a.x - b.x, a.y - b.y);
}

point operator * (const point &a, const ld &b) {
  return point(a.x * b, a.y * b);
}

point operator / (const point &a, const ld &b) {
  return point(a.x / b, a.y / b);
}

const ld EPS = 1e-8;
const ld INF = 1e20;
const ld PI = acosl(-1);

inline ld dist (point a, point b) {
  return hypotl(a.x - b.x, a.y - b.y);
}

inline ld sqDist (point a, point b) {
  return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) *
↪   (a.y - b.y);
}

inline ld dot (point a, point b) {
  return a.x * b.x + a.y * b.y;
}

inline ld cross (point a, point b) {
  return a.x * b.y - a.y * b.x;
}

inline ld cross (point a, point b, point c) {
  return cross(b - a, c - a);
}

inline point perp (point a) {
  return point(-a.y, a.x);
}

// circle through 3 points
pair <point, ld> getCircle (point a, point b, point
↪   c) {
  pair <point, ld> ret;
  ld den = (ld) 2 * cross(a, b, c);
  ret.x.x = ((c.y - a.y) * (dot(b, b) - dot(a, a))
↪   - (b.y - a.y) * (dot(c, c) - dot(a, a))) / den;
  ret.x.y = ((b.x - a.x) * (dot(c, c) - dot(a, a))
↪   - (c.x - a.x) * (dot(b, b) - dot(a, a))) / den;
  ret.y = dist(ret.x, a);
  return ret;
}

pair <point, ld> minCircleAux (vector <point> &s,
↪   point a, point b, int n) {
  ld lo = -INF, hi = INF;
  for (int i = 0; i < n; ++i) {
    auto si = cross(b - a, s[i] - a);
    if (fabs(si) < EPS) continue;
    point m = getCircle(a, b, s[i]).x;
    auto cr = cross(b - a, m - a);
    si < 0 ? hi = min(hi, cr) : lo = max(lo, cr);
  }
  ld v = 0 < lo ? lo : hi < 0 ? hi : 0;
  point c = (a + b) * 0.5 + perp(b - a) * v /
↪   sqDist(a, b);
  return {c, sqDist(a, c)};
}
```

```
pair <point, ld> minCircle (vector <point> &s, point a, int n) {
  random_shuffle(s.begin(), s.begin() + n);
  point b = s[0], c = (a + b) * 0.5;
  ld r = sqDist(a, c);
  for (int i = 1; i < n; ++i) {
    if (sqDist(s[i], c) > r * (1 + EPS)) {
      tie(c, r) = n == s.size() ? minCircle(s, s[i], i) : minCircleAux(s, a, s[i], i);
    }
  }
  return {c, r};
}

pair <point, ld> minCircle (vector <point> s) {
  assert(!s.empty());
  if (s.size() == 1) return {s[0], 0};
  return minCircle(s, s[0], s.size());
}

int n; vector <point> p;

int main() {
  cin >> n;
  while (n--) {
    double x, y;
    scanf("%lf %lf", &x, &y);
    p.emplace_back(x, y);
  }
  pair <point, ld> circ = minCircle(p);
  printf("%0.12f %0.12f %0.12f\n", (double) circ.x.x, (double) circ.x.y, (double) (0.5 * circ.y));
  return 0;
}
```

2.3 Point In Polygon

```
// Test if a point is inside a convex polygon in O(lg n) time
// Solves SPOJ INOROUT

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair <ll, ll> point;

#define x first
#define y second

struct segment {
  point P1, P2;

  segment () {}
  segment (point P1, point P2) : P1(P1), P2(P2) {}
};

inline ll ccw (point A, point B, point C) {
  return (B.x - A.x) * (C.y - A.y) - (C.x - A.x) * (B.y - A.y);
}

inline bool pointOnSegment (segment S, point P) {
  ll x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2 = S.P2.y;
  ll a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = a * c + b * d, len = c * c + d * d;
  if (x1 == x2 and y1 == y2) return x1 == x and y1 == y;
  if (dot < 0 or dot > len) return 0;
  return x1 * len + dot * c == x * len and y1 * len + dot * d == y * len;
}

const int M = 17;
const int N = 10010;

struct polygon {
  int n; // n > 1
  point p[N]; // clockwise order

  polygon () {}
  polygon (int _n, point *T) {
    n = _n;
    for (int i = 0; i < n; ++i) p[i] = T[i];
  }

  bool contains (point P, bool strictlyInside) {
    int lo = 1, hi = n - 1;
    while (lo < hi){
      int mid = lo + hi >> 1;
      if (ccw(p[0], P, p[mid]) > 0) lo = mid + 1;
      else hi = mid;
    }
    if (ccw(p[0], P, p[lo]) > 0) lo = 1;
    if (!strictlyInside and pointOnSegment(segment(p[0], p[n - 1]), P)) return 1;
    if (!strictlyInside and pointOnSegment(segment(p[lo], p[lo - 1]), P)) return 1;
    if (lo == 1 or ccw(p[0], P, p[n - 1]) == 0) return 0;
    return ccw(p[lo], P, p[lo - 1]) < 0;
  }
};

int q;
point P;
polygon p;

int main() {
  cin >> p.n >> q;
  for (int i = p.n - 1; i >= 0; --i) {
    scanf("%lld %lld", &p.p[i].x, &p.p[i].y);
  }
  while (q--) {
    scanf("%lld %lld", &P.x, &P.y);
    puts(p.contains(P, 0) ? "D" : "F");
  }
  return 0;
}
```

3 Graph

3.1 Dinic

```
// O(V^2 E), solves SPOJ FASTFLOW

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

struct edge {
  int u, v;
  ll cap, flow;
  edge () {}
  edge (int u, int v, ll cap) : u(u), v(v), cap(cap), flow(0) {}
};

struct Dinic {
  int N;
  vector <edge> E;
  vector <vector <int>> g;
  vector <int> d, pt;

  Dinic (int N) : N(N), E(0), g(N), d(N), pt(N) {}

  void AddEdge (int u, int v, ll cap) {
    if (u ^ v) {
      E.emplace_back(u, v, cap);
      g[u].emplace_back(E.size() - 1);
      E.emplace_back(v, u, 0);
      g[v].emplace_back(E.size() - 1);
    }
  }

  bool BFS (int S, int T) {
    queue <int> q({S});
    fill(d.begin(), d.end(), N + 1);
    d[S] = 0;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == T) break;
      for (int k : g[u]) {
        edge &e = E[k];
        if (e.flow < e.cap and d[e.v] > d[e.u] + 1) {
          d[e.v] = d[e.u] + 1;
          q.emplace(e.v);
        }
      }
    } return d[T] != N + 1;
  }

  ll DFS (int u, int T, ll flow = -1) {
    if (u == T or flow == 0) return flow;
    for (int &i = pt[u]; i < g[u].size(); ++i) {
      edge &e = E[g[u][i]];
      edge &oe = E[g[u][i] ^ 1];
      if (d[e.v] == d[e.u] + 1) {
        ll amt = e.cap - e.flow;
        if (flow != -1 and amt > flow) amt = flow;
        if (ll pushed = DFS(e.v, T, amt)) {
          e.flow += pushed;
          oe.flow -= pushed;
          return pushed;
        }
      }
    } return 0;
  }

  ll MaxFlow (int S, int T) {
    ll total = 0;
    while (BFS(S, T)) {
      fill(pt.begin(), pt.end(), 0);
      while (ll flow = DFS(S, T)) total += flow;
    }
    return total;
  }
};

int main() {
  int N, E;
  scanf("%d %d", &N, &E);
  Dinic dinic(N);
  for (int i = 0, u, v; i < E; ++i) {
    ll cap;
    scanf("%d %d %lld", &u, &v, &cap);
    dinic.AddEdge(u - 1, v - 1, cap);
    dinic.AddEdge(v - 1, u - 1, cap);
  }
  printf("%lld\n", dinic.MaxFlow(0, N - 1));
  return 0;
}
```

3.2 Eulerian Path

```
#include <bits/stdc++.h>

using namespace std;

// Eulerian path / circuit

// Undirected graph: circuit (or edge disjoint
↪  cycles) exists iff all nodes are of even degree
// Undirected graph: path exists iff number of odd
↪  degree nodes is zero or two

// Directed graph: circuit (or edge disjoint
↪  directed cycles) exists iff each node
//   satisfies in_degree = out_degree and the graph
↪  is strongly connected
// Directed graph: path exists iff at most one
↪  vertex has in_degree - out_degree = 1
//   and at most one vertex has out_degree -
↪  in_degree = 1 and all other vertices have
//   in_degree = out_degree, and graph is weakly
↪  connected

const int N = 200010;

bitset <N> bad;
vector <int> g[N];
vector <int> circ;
int n, m, deg[N], U[N], V[N];

void hierholzer (int src) {
  if (!deg[src]) return;
  vector <int> path;
  path.push_back(src);
  int at = src;
  while (!path.empty()) {
    if (deg[at]) {
      path.push_back(at);
      while (bad[g[at].back()]) g[at].pop_back();
      int e = g[at].back(), nxt = U[e] ^ at ^ V[e];
      bad[e] = 1, --deg[at], at = nxt; //change for
↪  directed
    } else {
      circ.push_back(at);
      at = path.back(), path.pop_back();
    }
  }
  reverse(circ.begin(), circ.end());
}

int main() {
  cin >> n >> m;
  for (int i = 1; i <= m; ++i) {
    scanf("%d %d", U + i, V + i);
    g[U[i]].push_back(i);
    g[V[i]].push_back(i); //change for directed
  }
  for(int i = 1; i <= n; i++) deg[i] = g[i].size();
  hierholzer(1); //change for directed [out(src) -
↪  in(src) = 1]
  for (int x : circ) printf("%d ", x); puts("");
  return 0;
}
```

3.3 Hopcroft Karp

```
#include <bits/stdc++.h>

using namespace std;

const int N = 40010;
const int INF = 1e8 + 5;

vector <int> g[N];
int n, m, p, match[N], dist[N];

bool bfs() {
  queue <int> q;
  for (int i = 1; i <= n; ++i) {
    if (!match[i]) dist[i] = 0, q.emplace(i);
    else dist[i] = INF;
  }
  dist[0] = INF;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    if (!u) continue;
    for (int v : g[u]) {
      if (dist[match[v]] == INF) {
        dist[match[v]] = dist[u] + 1,
        q.emplace(match[v]);
      }
    }
  }
  return dist[0] != INF;
}

bool dfs (int u) {
  if (!u) return 1;
  for (int v : g[u]) {
    if (dist[match[v]] == dist[u] + 1 and
↪  dfs(match[v])) {
      match[u] = v, match[v] = u;
      return 1;
    }
  }
  dist[u] = INF;
  return 0;
}

int hopcroftKarp() {
  int ret = 0;
  while (bfs()) {
    for (int i = 1; i <= n; ++i) {
      ret += !match[i] and dfs(i);
    }
  }
  return ret;
}

int main() {
  cin >> n >> m;
  // Bipartite Graph
  while (m--) {
    int u, v;
    scanf("%d %d", &u, &v);
    g[u].emplace_back(v);
    g[v].emplace_back(u);
  }
  // Maximum Matching, Minimum Vertex Cover
  int ans = hopcroftKarp();
  // Maximum Independent Set
  int offset = n - ans;
  cout << ans << " " << offset << '\n';
  return 0;
}
```

4 Math

4.1 Gaussian Elimination

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef long double ld;

const int N = 505;
const ld EPS = 1e-10;
const int MOD = 998244353;

ll bigMod (ll a, ll e, ll mod) {
  if (e == -1) e = mod - 2;
  ll ret = 1;
  while (e) {
    if (e & 1) ret = ret * a % mod;
    a = a * a % mod, e >>= 1;
  }
  return ret;
}

pair <int, ld> gaussJordan (int n, int m, ld
↪  eq[N][N], ld res[N]) {
  ld det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (fabs(eq[k][j])
↪  > fabs(eq[piv][j])) piv = k;
    if (fabs(eq[piv][j]) < EPS) continue; pos[j] =
↪  i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k],
↪  eq[i][k]);
    if (piv ^ i) det = -det; det *= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i) {
      ld x = eq[k][j] / eq[i][j];
      for (int l = j; l <= m; ++l) eq[k][l] -= x *
↪  eq[i][l];
```

```
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 :
↪   res[i] = eq[pos[i]][m] / eq[pos[i]][i];
  }
  for (int i = 0; i < n; ++i) {
    ld cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += eq[i][j] *
↪   res[j];
    if (fabs(cur) > EPS) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}

pair <int, int> gaussJordanModulo (int n, int m,
↪ int eq[N][N], int res[N], int mod) {
  int det = 1;
  vector <int> pos(m, -1);
  const ll mod_sq = (ll) mod * mod;
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j] >
↪   eq[piv][j]) piv = k;
    if (!eq[piv][j]) continue; pos[j] = i;
    for (int k = j; k <= m; ++k) swap(eq[piv][k],
↪   eq[i][k]);
    if (piv ^ i) det = det ? MOD - det : 0; det =
↪   (ll) det * eq[i][j] % MOD;
    for (int k = 0; k < n; ++k) if (k ^ i and
↪   eq[k][j]) {
      ll x = eq[k][j] * bigMod(eq[i][j], -1, mod) %
↪   mod;
      for (int l = j; l <= m; ++l) if (eq[i][l])
↪   eq[k][l] = (eq[k][l] + mod_sq - x * eq[i][l]) %
↪   mod;
    } ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 :
↪   res[i] = eq[pos[i]][m] * bigMod(eq[pos[i]][i],
↪   -1, mod) % mod;
  }
  for (int i = 0; i < n; ++i) {
    ll cur = -eq[i][m];
    for (int j = 0; j < m; ++j) cur += (ll)
↪   eq[i][j] * res[j], cur %= mod;
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}

pair <int, int> gaussJordanBit (int n, int m,
↪ bitset <N> eq[N], bitset <N> &res) {
  int det = 1;
  vector <int> pos(m, -1);
  for (int i = 0, j = 0; i < n and j < m; ++j) {
    int piv = i;
    for (int k = i; k < n; ++k) if (eq[k][j]) {
      piv = k; break;
    }
    if (!eq[piv][j]) continue; pos[j] = i,
↪   swap(eq[piv], eq[i]), det &= eq[i][j];
    for (int k = 0; k < n; ++k) if (k ^ i and
↪   eq[k][j]) eq[k] ^= eq[i]; ++i;
  }
  int free_var = 0;
  for (int i = 0; i < m; ++i) {
    pos[i] == -1 ? ++free_var, res[i] = det = 0 :
↪   res[i] = eq[pos[i]][m];
  }
  for (int i = 0; i < n; ++i) {
    int cur = eq[i][m];
    for (int j = 0; j < m; ++j) cur ^= eq[i][j] &
↪   res[j];
    if (cur) return make_pair(-1, det);
  }
  return make_pair(free_var, det);
}

int main() {

  return 0;
}
```

4.2 Pollard Rho

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;

namespace Rho {
  ull mul (ull a, ull b, ull mod) {
    ll ret = a * b - mod * (ull) (1.L / mod * a *
↪   b);
    return ret + mod * (ret < 0) - mod * (ret >=
↪   (ll) mod);
  }

  ull bigMod (ull a, ull e, ull mod) {
    ull ret = 1;
    while (e) {
      if (e & 1) ret = mul(ret, a, mod);
      a = mul(a, a, mod), e >>= 1;
    }
    return ret;
  }

  bool isPrime (ull n) {
    if (n < 2 or n % 6 % 4 != 1) return (n | 1) ==
↪   3;
    ull a[] = {2, 325, 9375, 28178, 450775,
↪   9780504, 1795265022};
    ull s = __builtin_ctzll(n - 1), d = n >> s;
    for (ull x : a) {
      ull p = bigMod(x % n, d, n), i = s;
      while (p != 1 and p != n - 1 and x % n and
↪   i--) p = mul(p, p, n);
      if (p != n - 1 and i != s) return 0;
    }
    return 1;
  }

  ull pollard (ull n) {
    auto f = [&] (ull x) {return mul(x, x, n) + 1;};
    ull x = 0, y = 0, t = 0, prod = 2, i = 1, q;
    while (t++ % 40 or __gcd(prod, n) == 1) {
      if (x == y) x = ++i, y = f(x);
      if ((q = mul(prod, max(x, y) - min(x, y),
↪   n))) prod = q;
      x = f(x), y = f(f(y));
    }
    return __gcd(prod, n);
  }

  vector <ull> factor (ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), r.begin(), r.end());
    return l;
  }
};

int t; ll n;

int main() {
  cin >> t;
  while (t--) {
    scanf("%lld", &n);
    vector <ull> facs = Rho::factor(n);
    sort(facs.begin(), facs.end());
    printf("%d", (int) facs.size());
    for (auto it : facs) printf(" %llu", it);
    puts("");
  }
  return 0;
}
```

5 Misc

5.1 Misc

```
// Pragmas
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx,avx2,fma")

// Custom Priority Queue
std::priority_queue< int, std::vector<int>,
↪ std::greater<int> > Q; // increasing

//gp hash table
↪ https://codeforces.com/blog/entry/60737
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::n
↪ ow().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^
↪   RANDOM; }
};
gp_hash_table<key, int, chash> table;

//bitset
BS._Find_first()
BS._Find_next(x) //Return first set bit after xth
↪ bit, x on failure
```

```
//Gray Code, G(0) = 000, G(1) = 001, G(2) = 011, G(3) = 010
inline int g(int n){ return n ^ (n >> 1); }

//Inverse Gray Code
int rev_g(int g) {
  int n = 0;
  for (; g; g >>= 1) n ^= g;
  return n;
}

/// Only for non-negative integers
/// Returns the immediate next number with same count of one bits, -1 on failure
long long hakmemItem175(long long n){
  if(!n) return -1;
  long long x = (n & -n);
  long long left = (x + n);
  long long right = ((n ^ left) / x) >> 2;
  long long res = (left | right);
  return res;
}

/// Returns the immediate previous number with same count of one bits, -1 on failure
long long lol(long long n){
  if(n < 2) return -1;
  long long res = ~hakmemItem175(~n);
  return (!res) ? -1 : res;
}

//Gilbert Ordering for Mo's Algorithm
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
  if (pow == 0) {
    return 0;
  }
  int hpow = 1 << (pow-1);
  int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) :
            ((y < hpow) ? 1 : 2);
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
  int64_t subSquareSize = int64_t(1) << (2*pow - 2);
  int64_t ans = seg * subSquareSize;
  int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
  return ans;
}

struct Query {
  int l, r, idx; // queries
  int64_t ord; // Gilbert order of a query
  // call query[i].calcOrder() to calculate the Gilbert orders
  inline void calcOrder() {
    ord = gilbertOrder(l, r, 21, 0);
  }
};
// sort the queries based on the Gilbert order
inline bool operator<(const Query &a, const Query &b) {
  return a.ord < b.ord;
}
```

6 String

6.1 Aho Corasick

```
#include <bits/stdc++.h>
using namespace std;

struct AC {
  int N, P;
  int A = 26;
  vector<vector<int>> next;
  vector<int> link, out_link;
  vector<vector<int>> out;

  AC() : N(0), P(0) {
    node();
  }

  int node() {
    next.emplace_back(A, 0);
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }

  inline int get(char c) {
    return c - 'a';
  }

  int add_pattern(const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][get(c)]) next[u][get(c)] = node();
      u = next[u][get(c)];
    }
    out[u].push_back(P);
    return P++;
  }

  void compute() {
    queue<int> q;
    for (q.push(0); !q.empty(); ) {
      int u = q.front();
      q.pop();
      for (int c = 0; c < A; ++c) {
        int v = next[u][c];
        if (!v) {next[u][c] = next[link[u]][c];}
        else {
          link[v] = u ? next[link[u]][c] : 0;
          out_link[v] =
            out[link[v]].empty() ? out_link[link[v]] : link[v];
          q.push(v);
        }
      }
    }
  }

  int advance(int u, char c) {
    while (u && !next[u][get(c)]) u = link[u];
    u = next[u][get(c)];
    return u;
  }

  void match(const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v]) {
        for (auto p : out[v]) cout << "match " << p << endl;
      }
    }
  }
};
// Don't forget to call compute()!

int main() {
  AC aho;
  int n;
  cin >> n;
  while (n--) {
    string s;
    cin >> s;
    aho.add_pattern(s);
  }
  aho.compute();
  string text;
  cin >> text;
  aho.match(text);
  return 0;
}
```

6.2 Palindromic Tree

```
#include <bits/stdc++.h>

using namespace std;

const int A = 26;
const int N = 300010;

char s[N]; long long ans;
int last, ptr, nxt[N][A], link[N], len[N], occ[N];

void feed (int at) {
  while (s[at - len[last] - 1] != s[at]) last = link[last];
  int ch = s[at] - 'a', temp = link[last];
  while (s[at - len[temp] - 1] != s[at]) temp = link[temp];
  if (!nxt[last][ch]) {
    nxt[last][ch] = ++ptr, len[ptr] = len[last] + 2;
    link[ptr] = len[ptr] == 1 ? 2 : nxt[temp][ch];
  }
  last = nxt[last][ch], ++occ[last];
}

int main() {
  len[1] = -1, len[2] = 0, link[1] = link[2] = 1, last = ptr = 2;
  scanf("%s", s + 1);
  for (int i = 1, n = strlen(s + 1); i <= n; ++i) feed(i);
  for (int i = ptr; i > 2; --i) ans = max(ans, len[i] * 1LL * occ[i]), occ[link[i]] += occ[i];
  printf("%lld\n", ans);
  return 0;
}
```

6.3 Suffix Array

```
/***
  * scan sa::str
  * n = strlen(sa::str)
  * call sa::build(n)

  * there are n+1 suffixes including the null suffix(denoted as n'th suffix, 0 based suffix indexing)
  * S[0 ... n] is the suffix array ( n+1 elements including the null suffix )
  * null suffix will be in the 0'th position of S
  * rnk[i] denotes the index of the i'th suffix in S[]
  * lcp[0] = 0, lcp[i] = longest commong prefix( suffix S[i-1], suffix S[i] )
***/

namespace sa {
  const int N = 100010; /// maximum possible string size

  char str[N];
  int wa[N], wb[N], wv[N], wc[N];
  int r[N], S[N], rnk[N], lcp[N];

  int cmp(int *r, int a, int b, int l) {
    return r[a] == r[b] && r[a + l] == r[b + l];
  }
  void da(int *r, int *sa, int n, int m) {
    int i, j, p, *x = wa, *y = wb, *t;
    for(i = 0; i < m; i++) wc[i] = 0;
    for(i = 0; i < n; i++) wc[x[i] = r[i]]++;
    for(i = 1; i < m; i++) wc[i] += wc[i - 1];
    for(i = n - 1; i >= 0; i--) S[--wc[x[i]]] = i;
    for(j = 1, p = 1; p < n; j *= 2, m = p) {
      for(p = 0, i = n - j; i < n; i++) y[p++] = i;
      for(i = 0; i < n; i++) if(S[i] >= j) y[p++] = S[i] - j;
      for(i = 0; i < n; i++) wv[i] = x[y[i]];
      for(i = 0; i < m; i++) wc[i] = 0;
      for(i = 0; i < n; i++) wc[wv[i]] ++;
      for(i = 1; i < m; i++) wc[i] += wc[i - 1];
      for(i = n - 1; i >= 0; i--) S[--wc[wv[i]]] = y[i];
      for(t = x, x = y, y = t, p = 1, x[S[0]] = 0, i = 1; i < n; i++) x[S[i]]= cmp(y, S[i - 1], S[i], j) ? p - 1 : p++;
    }
  }
  void calheight(int *r, int *sa, int n) {
    int i, j, k = 0;
    for(i = 1; i <= n; i++) rnk[S[i]] = i;
    for(i = 0; i < n; lcp[rnk[i++]] = k ) {
      for(k ? k-- : 0, j = S[rnk[i]-1]; r[i+k] == r[j+k]; k++);
    }
  }
  void build(int n) {
    for(int i = 0; str[i]; i++) r[i] = (int)str[i]; // or do some scaling
    r[n] = 0;
    da(r, S, n+1, 128); // 128 -> maximum possible asci value of a character + 1
    calheight(r, S, n);
  }
}
```

3

6.4 Z Algorithm

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
char s[N];
int t, n, z[N];
int main() {
  scanf("%s", s);
  n = strlen(s), z[0] = n;
  int L = 0, R = 0;
  for (int i = 1; i < n; ++i) {
    if (i > R) {
      L = R = i;
      while (R < n && s[R - L] == s[R]) ++R;
      z[i] = R - L; --R;
    } else {
      int k = i - L;
      if (z[k] < R - i + 1) z[i] = z[k];
      else {
        L = i;
        while (R < n && s[R - L] == s[R]) ++R;
        z[i] = R - L; --R;
      }
    }
  }
  for (int i = 0; i < n; ++i) {
    printf("%d --> %d\n", i, z[i]);
  }
  return 0;
}
```